In the Claims:

Please amend claims 1-30, 36, 38-40, and 43, as indicated below.

1. (Currently amended) A computer readable storage medium encoding program code executable on one or more processors to implement:

instantiating [[of]] a data structure implementation in a memory~~, the encoding~~ comprising[[:]] ~~a definition of~~ a double-ended array ~~instantiable in memory~~; ~~and~~

~~a functional encoding of~~ executing a plurality of opposing-end access operations that, when executed on ~~respective~~ the one or more processors~~,~~ ~~that~~ access the memory, and provide concurrent push-type and pop-type access [[at]] to at least one of the opposing ends and concurrent, opposing-end accesses that are non-interfering for at least some states of the array[[,]]; and

mediating concurrent execution of the access operations ~~is mediated~~ using a single-target synchronization primitive;

wherein the data structure implementation is linearizable and non-blocking, and

~~wherein concurrent execution of the access operations is mediated using a single-target synchronization primitive,~~

wherein the single-target of the single-target synchronization primitive includes a value encoding for an element of the array and a version number encoded integrally therewith.

2. (Currently amended) The ~~data structure encoding~~ storage medium of claim 1,

wherein the concurrent opposing-end access operations are non-interfering for all but boundary condition states of the array.

3. (Currently amended) The ~~data structure encoding~~ storage medium of claim 1, wherein the non-blocking implementation is obstruction-free, though not wait-free or lock-free.

4. (Currently amended) The ~~data structure encoding~~ storage medium of claim 1, wherein the single-target synchronization primitive employs a Compare-And-Swap (CAS) operation.

5. (Currently amended) The ~~data structure encoding~~ storage medium of claim 1, wherein the single-target synchronization primitive employs a Load-Linked (LL) and Store-Conditional (SC) operation pair.

6. (Currently amended) The ~~data structure encoding~~ storage medium of claim [[4]] 1, wherein said mediating comprises attempting to increment the version number included in the single-target of the single-target synchronization primitive ~~includes a value encoding for an element of the array and a version number encoded integrally therewith~~.

7. (Currently amended) The ~~data structure encoding~~ storage medium of claim 1, wherein the double-ended array implements a deque.

8. (Currently amended) The ~~data structure encoding~~ storage medium of claim 1, wherein the opposing-end access operations are at least consistent with semantics of a FIFO queue.

9. (Currently amended) The ~~data structure encoding~~ storage medium of claim 1, wherein the boundary-condition states include an empty state.

10. (Currently amended) The ~~data structure encoding~~ storage medium of claim 1, wherein the boundary-condition states include a single element state.

11. (Currently amended) The ~~data structure encoding~~ storage medium of claim 1, wherein the array is indexable as a circular array.

12. (Currently amended) The ~~data structure encoding~~ storage medium of claim 11,

wherein the boundary-condition states include a full state.

13. (Currently amended) The ~~data structure encoding~~ storage medium of claim 11,

wherein the opposing-end accesses include opposing-end, push-type accesses; and

wherein the boundary-condition states include a nearly full state.

14. (Currently amended) The ~~data structure encoding~~ storage medium of claim 1, wherein distinct left null and right null distinguishing values are employed to identify free elements of the array.

15. (Currently amended) The ~~data structure encoding~~ storage medium of claim 14,

wherein the array is indexed as a circular array; and

wherein an additional distinguishing value is employed to facilitate consumption of free elements by push-type operations at either end of the array.

16. (Currently amended) The ~~data structure encoding~~ storage medium of claim 1, embodied as a software component combinable with program code to provide the program code with non-blocking access to a concurrent shared object.

17. (Currently amended) The ~~data structure encoding~~ storage medium of claim 1,

~~embodied as a~~ wherein the program code is further executable to provide non-blocking access to a concurrent shared object.

18. (Currently amended) The ~~data structure encoding~~ storage medium of claim 1, ~~wherein the computer readable medium includes~~ comprising at least one medium selected from the set of a disk, tape or other magnetic, optical, or electronic storage medium ~~and a network, wireline, wireless or other communications medium~~.

19. (Currently amended) A computer readable storage medium encoding program code executable on one or more processors to implement:

a single-target synchronization primitive based, non-blocking, fully functional deque implementation for which concurrent opposing-end access operations do not always interfere, ~~and~~

wherein shared storage usage of the deque implementation is insensitive to a number of access operations that concurrently access the deque.

20. (Currently amended) The ~~CAS-based non-blocking deque implementation~~ storage medium of claim 19,

wherein the deque implementation is obstruction-free, though not wait-free or lock-free.

21. (Currently amended) The ~~non-blocking deque implementation~~ storage medium of claim 19,

wherein the concurrent opposing-end access operations are non-interfering for all but boundary condition states.

22. (Currently amended) The ~~non-blocking deque implementation~~ storage medium of claim 19,

wherein state of the deque is encoded using an array.

23. (Currently amended) The ~~non-blocking deque implementation~~ storage medium of claim 22,

wherein the array is a circular array.

24. (Currently amended) The ~~non-blocking deque implementation~~ storage medium of claim 19,

wherein the single-target synchronization includes use of a Compare-And-Swap (CAS) operation.

25. (Currently amended) The ~~non-blocking deque implementation~~ storage medium of claim 19,

wherein the single-target synchronization includes use of a Load-Linked (LL) and Store-Conditional (SC) operation pair.

26. (Currently amended) The ~~non-blocking deque implementation~~ storage medium of claim 19,

wherein at least some concurrently executed access operations interfere with each other; and

wherein the interfering concurrently executed access operations are each retried.

27. (Currently amended) The ~~non-blocking deque implementation~~ storage medium of claim 26,

wherein the non-blocking deque implementation does not guarantee that at least one of the interfering concurrently executed access operations makes progress.

28: (Currently amended) The ~~non-blocking deque implementation~~ storage medium of claim 27,

wherein a separate contention management facility is employed to ensure progress in a concurrent computation that employs the deque implementation.

29. (Currently amended) The ~~non-blocking deque implementation~~ storage medium of claim 19,

~~embodied as a software that~~ wherein the program code is executable to implement: ~~defines a representation of~~

instantiating the deque ~~instantiable~~ in memory; and ~~which includes a functional encoding of~~

executing access operations ~~executable by one or more processors~~ to operate on state of the deque.

30. (Currently amended) A method of managing obstruction-free access to a shared double-ended array, the method comprising:

instantiating the double-ended array in memory; and

operating on state of the array using access operations that detect interference by other executions thereof using a single-target synchronization primitive; and

after detection of an interfering execution, retrying an interfered-with access operation,

wherein execution of respective ones of the access operations allow[[s]] at least (i) concurrent push-type and pop-type access [[at]] to at least one of the opposing ends and (ii) concurrent, opposing-end accesses that are non-interfering for at least some states of the array, and

wherein the single-target of the single-target synchronization primitive includes a value encoding for an element of the array and a version number encoded integrally therewith.

31. (Original) The method of claim 30,
wherein the concurrent, opposing-end accesses are non-interfering for all but boundary-condition states of the array.

32. (Original) The method of claim 30,
wherein execution of the access operations is obstruction-free, though not wait-free or lock-free.

33. (Original) The method of claim 30,
wherein the single-target synchronization primitive employs a Compare-And-Swap (CAS) operation.

34. (Original) The method of claim 30,
wherein the single-target synchronization primitive employs a Load-Linked (LL) and Store-Conditional (SC) operation pair.

35. (Original) The method of claim 30,
wherein the double-ended array includes a representation of a deque; and
wherein the access operations include both push-type and pop-type access operations at both opposing ends of the deque.

36. (Currently amended) The method of claim 30, further comprising:
~~wherein~~ a contention management facility ~~facilitates~~ facilitating progress of access operations.

37. (Original) The method of claim 36, further comprising:

changing, during the course of a computation involving the shared double-ended array, a contention management strategy employed by the contention management facility.

38. (Currently amended) The method of claim 36, ~~further comprising:~~

~~operating~~ wherein the ~~separate~~ contention management facility is separable from the single-target synchronization primitive.

39. (Currently amended) The method of claim 30,

wherein progress is ensured not by the shared double-ended array ~~object implementation~~, but rather by a separate contention management facility.

40. (Currently amended) An apparatus, comprising:

one or more processors;

one or more data stores addressable by each of the one or more processors; and

means for coordinating concurrent non-blocking execution, by one or more of the processors, of at least opposing-end push-type and pop-type access operations on a fully functional deque data structure encoded in the one or more data stores, the coordinating employing a compare-and-swap (CAS) synchronization primitive to detect interference of concurrently executed ones of the access operations, the coordinating means ensuring that, for all but boundary-condition states of the deque, opposing-end accesses are non-interfering, wherein the target of the CAS synchronization primitive includes a value encoding for an element of the deque and a version number encoded integrally therewith.

41. (Original) The apparatus of claim 40,

wherein the coordinating means tolerates non-progress of interfering executions of the access operations.

42. (Original) The apparatus of claim 40, further comprising:

means for managing contention between interfering executions of the access operations.

43. (Currently amended) A non-blocking method of operating on a double-ended queue data structure, the method comprising:

concurrently executing push-type and pop-type access operations [[at]] to at least one of opposing ends of the double-ended queue;

detecting interference with a particular execution of one of the access operations using a single-target synchronization primitive; and

tolerating, in the implementation of the double-ended queue data structure, a possibility that two or more executions of the access operations interfere with each other and each consequently fail to make progress,

wherein the non-blocking property is achieved while ensuring that, for all but boundary-condition states of the deque, opposing-end accesses are non-interfering and without use of a multi-target synchronization primitive, and

wherein the single-target of the single-target synchronization primitive includes a value encoding for an element of the double-ended queue and a version number encoded integrally therewith.

44. (Original) The method of claim 43, further comprising:

managing the possibility that access operations interfere with each other and consequently fail to make progress using a substitutable contention management facility separable from implementation of the double-ended queue data structure.